DOCUMENT RESUME

ED 341 557 SE 052 485

AUTHOR Greenwell, Raymond N.
TITLE Problem Solving via Pascal, with Data Structures. Dissemination Packet--Summer 1989: Booklet #4.
INSTITUTION Hofstra Univ., Hempstead, NY. Dept. of Mathematics.; Hofstra Univ., Hempstead, NY. School of Secondary Education.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE 89
CONTRACT TEI8550088,8741127
NOTE 52p.; For related documents, see SE 052 482-490.
PUB TYPE Guides - Classroom Use - Teaching Guides (For Teacher) (052) -- Computer Programs (101) -- Tests/Evaluation Instruments (160)

EDRS PRICE MF01/PC03 Plus Postage.
DESCRIPTORS *Computer Assisted Instruction; *Computer Software Evaluation; Higher Education; High Schools; *Inservice Teacher Education; *Mathematics Education; *Mathematics Teachers; Program Descriptions; Secondary School Mathematics; Secondary School Teachers; Teacher Education Programs; Teacher Workshops
IDENTIFIERS *Hofstra University NY; *PASCAL Programing Language

ABSTRACT

This booklet is the fourth in a series of nine from the Teacher Training Institute at Hofstra University (New York) and provides descriptive information about the introductory course in Pascal programing with emphasis on the solving of problems found in the advanced-placement computer science curriculum of secondary school mathematics. Included in this booklet are: (1) the instructor's evaluation of the behavioral aspects and affective observations gleaned from his 3 years of participation in this program, as well as proposals for program improvement; (2) a short appraisal of the program and comments from one participant; (3) the course outlines for each year; (4) a sampler of homework assignments, class notes, and computer programs used in the courses; and (5) the examinations used in the courses with some handwritten solutions. An extensive sample of the instructor's and the participants' course project solutions using Pascal programs can be found in booklet #5 in this series (SE 052 486). (JJK)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

SE

ED341557

HOFSTRA UNIVERSITY

TEACHER TRAINING INSTITUTE

Department of Mathematics and School of Secondary Education
Hofstra University
Hempstead, NY 11550

DISSEMINATION PACKET - SUMMER 1989

Booklet #4

RAYMOND N. GREENWELL

PROBLEM SOLVING VIA PASCAL, WITH DATA STRUCTURES

NSF Grant # TEI8550088, 8741127

SE052485

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☒ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Raymond Greenwell

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

BEST COPY AVAILABLE

This booklet is the fourth in a series of nine booklets which constitute the Hofstra University Teacher Training Institute (TTI) packet. The Institute was a National Science Foundation supported three-year program for exemplary secondary school mathematics teachers. Its purpose was to broaden and update the backgrounds its participants with courses and special events and to train and support them in preparing and delivering dissemination activities among their peers so that the Institute's effects would be multiplied.

This packet of booklets describes the goals, development, structure, content, successes and failures of the Institute. We expect it to be of interest and use to mathematics educators preparing their own teacher training programs and to teachers and students of mathematics exploring the many content areas described.

"Problem Solving via Pascal" was an introductory course in Pascal programming, with an emphasis on solving problems related to the high school mathematics curriculum. "Problem Solving via Pascal Data Structures" was a followup course covering those topics in the Advanced Placement Computer Science curriculum not discussed in the first course.

This booklet gives the syllabi and exams for both Pascal courses along with a complete set of class handouts. Also included are the instructor's evaluation of the course and a

participant's comments.

Report on the Teacher Training Institute
Math 287: Problem Solving Through Pascal
and
Math 299A: Problem Solving Using Pascal Data Structures

Booklet #4

Raymond N. Greenwell
Department of Mathematics
Hofstra University
Hempstead, NY 11550

copyright (c) 1989 Raymond N. Greenwell
All rights reserved, except that permission will be granted to make a limited number of copies for noncommercial educational purposes, upon written request, provided that this copyright notice shall appear on all such copies.

Table of Contents

A. Instructor's Evaluations

B. A Participant's Comments

C. Syllabi

D. A Sampler of Handouts: Exercises, Notes and Programs

E. Exams (some with handwritten solutions)

Booklet #5 contains a sampler of instructor and participant project solutions.

Report on the Teacher Training Institute
Math 287: Problem Solving Through Pascal

by
Raymond N. Greenwell
Associate Professor
Department of Mathematics

Mathematics 287 (Problem Solving Through Pascal) was offered in Fall of 1987 as part of the first round of the Teacher Training Institute. The class met every Thursday afternoon from 4:20 to 6:45, with a break of about 15 minutes. The course was successful in many of its objectives, but there were also problems.

The success of the class could be measured partly through comparing the pre- and post-test scores. Professor Esin Kaya of the Hofstra University Department of Education can give the exact figures, but I can state in general terms that most of the participants got most of the questions wrong on the pre-test, while almost everyone achieved a perfect score on the post-test. They clearly learned a lot.

More information was obtained from the evaluations the participants filled out, as well as comments they made informally. Many of the participants were delighted with the course. They learned things they wanted to learn, made new professional contacts, and became better equipped to teach some of the courses they were teaching. On the other hand, many of the participants were unhappy with the course. They found the material too difficult and the assignments too demanding of their time. For them, the course was a grueling experience. I will address some of the reasons why, in my opinion, this occurred.

One of the first problems is that the program in general, and my course in particular, was not planned well enough. This was not necessarily the fault of the planners; the problem was that we hadn't done this before and could not know exactly how things would work out. In the original plan, all participants in the course would be able to write simple Pascal programs with output on paper by the end of the summer. This was to occur either through their prior experience, through work done in one of the summer courses, or through working outside the class during the summer. When we met in the fall, I intended to continue from where the prior training left off. Unfortunately, I discovered in our initial fall meeting that many of the participants knew nothing of Pascal or of how to write a program. The summer had been too busy for them to learn anything beyond what was

required for their summer courses in the Institute. Further, some of those who had credentials indicating they could program, in fact, could not. This was often because their background was shallow. They may have taken some computer courses, but if insufficient work was demanded in those courses, the material never stayed with them. Thus my expectations and theirs were at odds on the first day. I spent time backtracking, trying to bring them up to where they should be, but a few never quite got over the feeling that I was expecting them to do something that was, in their eyes, impossible.

These feelings were further aggravated by my expectations that, since this was a select group of high school mathematics teachers, they should be better at solving problems than the typical group of undergraduates. What I discovered as the semester went on is that, although they may be considered good teachers, some were poor students. This was often because their background was weaker than it appeared at first. Some of them had degrees in mathematics education in which they did very little serious mathematics. For example, when I gave a problem that required knowledge of trigonometry, I assumed this was a topic all high school teachers knew very well. I was wrong. It came as a shock to me that some high school teachers know less about trigonometry that what I expect my freshman calculus students to know. For other participants, it seemed that the pace of a college course was too fast. After years of teaching high school, some were used to slow going, with lots of review, and not much expected outside of class. I tried to remind them that college is not like that, but they pointed out that college students don't have families to take care of and fulltime jobs as teachers. Since all of us teaching in the program had been telling the participants what an honor it was for them to be here and what wonderful teachers they were, they were not ready to be told that their performance as students was inadequate.

As a result, even some of those who learned a lot from the course and did good work felt sour about the experience. On the other hand, some of the truly excellent teachers enjoyed the experience and felt apologetic about the complaints and bad attitudes of their fellow teachers. Despite the successes, it was not an enjoyable course for many of the participants and myself alike.

Changes made in Round Two

In round two of the Teacher Training Institute, I taught Math 287 three afternoons a week for two hours an afternoon during the five-week summer session, plus once a month during the fall. Although the participants seemed

less qualified on paper, their performance and attitude was superior to those of the round one participants. I judged it to be a far greater success than the round one course. This is due to changes made to overcome the mistakes of round one.

First, a pre-course was held the last week of June to introduce the novices to writing short programs in Pascal and getting the results on paper. This only required four sessions of three hours each, but when the regular course started, I knew that everyone had at least a minimum background. This allowed me to teach the course at a satisfactory level, without boring the experienced or terrifying the beginners.

Second, by running the course through the summer and the fall, the participants had more time to get their programs working. Computer programming takes time, and the first round participants often couldn't find the time between one week and the next to get on the computer. Spreading the programs out made all the difference in the world. Not only did almost all participants get their programs working on time, but many found time to add extra credit features to their programs. In addition, the programming assignments in round two were harder, requiring more thought and time.

Third, my attitude had changed. I now knew what I was working with, and I came down a lot harder on what I expected. I did not present the course as an honor just for them to be there; I told them from the beginning that programming takes time, and that they should plan their schedules accordingly. When I got to the part where I wanted to use trigonometry, I told them that all high school teachers should know this material, but some may not, so I would spend a short time reviewing it. Those for whom this review was insufficient would need to spend time outside of class, not only because it was going to be on the test, but because they had no business being high school math teachers if they didn't know trigonometry. That worked. One teacher borrowed a precalculus text from me; others dug up their high school math texts. And they did learn trigonometry.

Fourth, I had a much clearer conception of the course. In round one, my plan to integrate mathematical problem solving with learning Pascal started out somewhat vague and evolved over time. In round two, I taught a more traditional introduction to computer science, much like the first part of A.P. course in computer science.

The results of the class impressed me. In fact, I have only once or twice before taught a class whose performance was as superb as the performance

of these teachers. The class was a joy to teach. And although the participants still found the class difficult and challenging, they enjoyed the experience of learning.

One problem arose in the fall, when the teachers only saw me once a month. Many felt they forgot a lot in between sessions and had a hard time getting back into the swing of things. This problem was most acute at the final examination in December, when many teachers felt they did poorly because it had been so long since they had been immersed in the material. Nevertheless, their scores on the final exam were not as bad as they had expected. The class average was a respectable 76%. The teachers' perception of having forgotton most of what they learned did not seem born out by their performance.

The Coda: Math 299A

In summer of 1988, I taught "Problem Solving Using Pascal Data Structures," which was intended as a follow-up course to Math 287. My goal was to familiarize the teachers with the topics in the A.P. Computer Science Curriculum which I had not covered in Math 287: pointers, linked lists, queues, searching, and sorting. The course also included trees and stacks, which I had discussed briefly in the first round of Math 287. I wasn't bothered by this small amount of overlap, especially since only one of the teachers in the course had taken Math 287 in round one. The rest were from round two, except for two teachers new to the program.

The course met for eleven two hour sessions over four weeks. This is not really enough time to do data structures in depth, so the course deliberately was superficial. Also, with only three weekends, there was less time outside of class to work on projects. As a result, only two-thirds of the class had completed all the projects by the time of the final exam. And although the post-test scores showed a great improvement over the pre-test scores, the class average for the final exam was only 68%. Given the constraints, and the fact that many of the participants were taking a second class over the four weeks, this may be as good as can be expected.

Proposal for a third round:

There is still a need for high school teachers to know more about problem solving and using the language Pascal. Before running the program again, however, I would change several features that I see as problems.

First, the program as it currently exists is difficult enough to deter many who could benefit from it. Some teachers are unwilling or unable to make a commitment for a summer plus an academic year. Even those who may consider participating for the summer alone may not be prepared to take two intensive courses over five weeks. With one course in the morning and one in the afternoon, they are left with little time to enjoy and explore the material, to say nothing of living a normal life.

Second, many high school teachers need to work during the summer and can't afford to enroll in a program which provides no financial remuneration.

Thus, I propose a program consisting of two summers. In the first summer, participants would take Math 287 alone for five weeks. The course would meet four mornings a week, leaving the afternoons free to work on solving problems and working on programs. The following summer the participants would take a more complete and thorough version of Math 299A. Participants who finish the two courses would be prepared to teach the A.P. Computer Science curriculum. Some participants may choose to stop after the first course, but other participants with stronger backgrounds in Pascal may choose to omit the first course and take only the second.

In between the two summers, the participants would complete a project at their own school. Typical projects would be teaching a Pascal course which they hadn't taught before, bringing new ideas, concepts, and activities into a Pascal course they had taught before, or training other teachers at their schools about what they learned in the course.

During each of the two summers, the teachers would receive a stipend to help replace lost earnings.

Student Commentary on the Teacher Training Institute
Math 287: Problem Solving Through Pascal
Math 299A: Coda (Pascal Data Structures)

After reviewing Dr. Raymend Greenwell's report on the three courses he offered in Pascal for the Teacher Training Institute, I feel the need to identify myself before I add any further comments. I was a Cycle II participant as well as a student in the summer coda program. I had a "mild" background in Pascal before entering any of Dr, Greenwell's courses, having taken a graduate level introductory course in Pascal and having taught a one-half sememster course in Pascal in the high school.

The underlying concerns I feel Dr. Greenwell expressed, namely that the teacher participants did not have a strong enough mathematics background and that there was a great feeling of being overwhelmed are correct. However, these need to be addressed (defended?) separately. Several of the Cycle II participants were indeed Junior High teachers or even not directly in a classroom setting. When you spend many years doing rememdial work of this nature or do not use the more advanced concepts, they are not at your finger-tips. Mathematics is somewhat like a language that gets rusty when not used. Granted too, there were some participants whose mathematical background truly was lacking, but these people should have been screened by the directors of the program and perhaps be encouraged not to take this course if its goals were so high. I always feel however, that you cannot please all of the people all of the time.

The feeling of being overwhelmed was, I think more of a concern. Naturally those who were struggling mathematically would be struggling with the programming assignments too as the two are so inter-related. However, those of us who were "mathematically sound" often needed more hours than were in the day to keep up with the reading and programming assignments. Of course there were exams also to be considered. Most of us were simultaneously taking other courses, not to mention that during the summer there are family obligations and during the school year there are teaching responsibilities which could not be pushed aside. (I often found myself working through until 2 a.m. on the assignments!)

Despite all of the hours of hard work, I feel the courses I participated in were highly successful and perhaps Dr. Greenwell was too hard on himself in his commentary. We learned, we grumbled, but we grew in our Pascal abilities. If I were asked to assess the courses, I would rate them as highly succesful....the measure of success depending on the individual's input, seriousness of purpose and his/her desire to achieve.

Respectfully submitted by
Irene Ober

April 13, 1989

Mathematics 287: Problem Solving Through Pascal
Fall 1986
100 South Hall, Thursday 4:20-6:45
Instructor: Dr. Raymond Greenwall
Office: 104 South Hall, 560-5573
Texts: *How to Solve It by Computer*, R. G. Dromey
Pascal for Programmers, Lecarme and Nebut
Your Own Favorite Pascal Text, by Hu Meyer

There will be five computer assignments, worth 40 points each, plus one or two other non-computer exercises or short computer projects. There will also be a midterm and a final exam, both worth roughly 130 points.

Schedule

Sept. 11	Preliminaries
Sept. 18	How to write a program
Sept. 25	1 dimensional arrays
Oct. 2	Procedures, functions, and parameters
Oct. 9	2 dimensional arrays and Boolean variables
Oct. 16	Mathematical induction
Oct. 23	Records
Oct. 30	**Midterm exam**
Nov. 6	Stacks
Nov. 13	Infix, prefix, and postfix notation; sets
Nov. 20	Recursion
Dec. 4	Special session--to be announced
Dec. 11	Review
Dec. 18	**Final exam**

Mathematics 287: Problem Solving Through Pascal
Summer-Fall 1987
100 South Hall
Instructor: Dr. Raymond Greenwell
Office: 104 South Hall, 560-5573
Texts: *Oh! Pascal!* (2nd ed), Doug Cooper and Michael Clancy

There will be roughly six computer assignments, worth 30 points each, as well as a midterm and a final exam, both worth roughly 130 points.

Schedule

Preliminary course (for those who need it):

Chap. 1 & 2 Simple programs, constants, variables, reading and writing, arithmetic expressions, standard functions, getting copies of programs and output on paper.

Summer Session

Week 1 Chap. 3 & 4: procedures, functions, **for** statement.
Week 2 Chap. 5 & 6: case and if statements.
Week 3 Chap. 7 & 8: **while** loops, recursion, text processing.
Week 4 Chap. 9 & 10: Ordinal types, MIDTERM.
Week 5 Chap. 11 & 12: arrays and records.

Fall

Chap. 13, 14, & (with a little luck) 15
Files, sets, and (here's the luck part) pointers.

Mathematics 299A: Problem Solving via Pascal Data Structures
Summer 1988
100 South Hall
Instructor: Dr. Raymond Greenwell
Office: 104 South Hall, 560-5573
Text: *Pascal Plus Data Structures, Algorithms, and Advanced Programming* (2nd ed.) by Nell Dale and Susan Lilly

The prerequisite for this course is a knowledge of Pascal through arrays, records, procedures, and recursion. The course covers the part of the AP Computer Science curriculum commonly referred to as "data structures." The grade will be based on roughly four computer assignments and a final exam.

Independent reading:
Chap. 1: Programming Tools
Chap. 2: Verifying, Debugging, and Testing (Don't worry about the details of the application on pp. 67-83).

Tentative Schedule

July 5	Chap. 3: Data Design
July 7	Chap. 4: Stacks
July 11	Chap. 5: FIFO Queues
July 12	Chap. 6: Linked Lists
July 14	Chap. 6 (continued)
July 18	Chap. 7: More Linked Lists
July 19	Chap. 9: Binary Search Trees
July 21	Chap. 10: Binary Expression Trees, Heaps, and Graphs
July 25	Chap. 11: Sorting
July 26	Chap. 12: Searching
July 28	Final Exam

SQUARE-SERVICE CORPORATION
Service: To compute n^2 for the customer's non-negative integer n.

Job specification to employee:
1) Get a number from your boss and name it x.
2) If x is 0, then wake up your boss, return the value of x to him or her, and stop.
3) If x is not 0, then hire an assistant and give him or her a copy of the x - 1.
4) Go to sleep until your assistant wakes you up.
5) When your assistant wakes you up, get the number he or she returns to you, name it y, and fire him or her.
6) Compute y + x + x - 1, wake up your boss, give him or her a copy of this number, and stop.

FIBONACCI SERVICE CORPORATION
Service: To compute the Fibonacci number F(x) for the customer's non-negative integer x, where

$$F(x) = \begin{cases} 1 & \text{if } x = 0 \text{ or } x = 1 \\ F(x-1) + F(x-2) & \text{if } x > 1. \end{cases}$$

Job specification to employee:
1) Get a number x from your boss.
2) If x is less than 2, then return 1 to your boss and stop.
3) Otherwise hire two assistants, and give one assistant a copy of the number x - 1 and the other assistant a copy of the number x - 2.
4) Take a nap while your assistants are working.
5) When your assistants wake you, get a number, y, from assistant one, and a number, z, from assistant two. Fire your assistants.
6) Compute y + z, wake your boss, and give him or her a copy of this result. Then stop.

Exercises for recursion:

1) Write a job specification for the NFACTR Service Corporation, which computes n!.

2) Euclid's algorithm for finding the greatest common divisor of two integers is defined by:

$$GCD(m, n) = \begin{cases} GCD(n, m) & \text{if } n > m \\ m & \text{if } n = 0 \\ GCD(n, m \bmod n) & \text{otherwise} \end{cases}$$

where m mod n is the remainder when m is divided by n. Write a Pascal program to calculate GCD(m, n), and then trace through the program to find GCD(6, 20) and GCD(60, 105). Use the built-in Pascal function mod.

3) Ackerman's function is defined recursively on the nonnegative integers as follows:

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m \neq 0, n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m \neq 0, n \neq 0 \end{cases}$$

Write a Pascal program to calculate A(m, n) and trace through it to show that A(2, 2) = 7.

Math 287 Programming Project 1 due _____
Grade Point Average

Grades at Hofstra University are awarded on the following basis: A = 4.0, B = 3.0, C = 2.0, D = 1.0, and F = 0.0. Letter grades can also be modified with + (plus 0.3 point) or - (minus 0.3 point). For example, A- counts as 3.7 points, and B+ is 3.3 points. There is no A+. If a professor gives such a grade, it only counts as 4.0 points. Similarly, an F+ or F- is worth 0.0 points, and a D- is worth 1.0 points.

Write a program that computes a grade-point average based on as many letter grades as the user wants to enter. Your program should:

a) explain what's going to happen, and ask the user how many grades will be entered;

b) compute the number of grade points corresponding to each letter grade entered (one per line), and keep a running total of the grade points;

c) print the user's grade point average--the total divided by the number of grades entered.

Assume that every letter grade is followed by a '+', a '-', or a space. Be sure to deal with the special cases such as A+ or F-.

You are to turn in:

a) a listing of your program, with proper comments, meaningful variable names, nice indentation, etc.

b) The output for the following sets of data:

i) 3 grades: A, B-, A-

ii) 4 grades: A+, B+, C, F+

iii) 4 grades: D-, B+, B, B-

Hints:

Write the program in stages. First write a function that converts the letter grade into a numerical grade. Put this function into a short program that inputs a single letter grade and then uses the function to print the numerical value.

Once you have this working, complete the main program so it will enter as many grades as the user desires and computes the grade point average.

For the experienced programmers:

The program as described counts all grades equally. Modify the program so that it asks how many credits each grade is worth and uses this in computing the grade point average.

Do some error checking. Modify the program so that grades such as E° will not be accepted. Modify it so the letter grade will be counted whether it is entered upper case or lower case.

Allow the user to type a single letter, such as A, without being required to type a blank following the letter.

You can probably think of some other improvements. Go ahead! Be sure to note on your program and/or output what you have accomplished.

Math 287 Project 3 due _____
Extended precision arithmetic

For this project you are to write a program that performs as a calculator with an arbitrarily large number of digits of accuracy. The basic project need only do integer addition up to 100 digits, but there is **far more to do for the adventuresome.**

The program should allow you to enter the first term, digit by digit, followed by a '+', followed by the second term, followed by an '='. It should then print out the correct result. The program should allow you to continue doing this until you type 'q'.

Run the program with the following computations:

1) 387426865+678=
2) 678+387426865=
3) 94327068518+12305678944=
4) 9999+11=

The first operation verifies that your program can add a small number to a large number. The second verifies that the small number can be added first. The third verifies that two large numbers can be added, with the result having a greater number of digits than either term. The fourth verifies that, after adding large numbers, the program can go back to adding small numbers.

The program should be reasonably efficient so that it only adds nonzero digits. In other words, it should not add up 100 digits when your numbers are only 2 digits long.

Extra features for the experienced to try:

Allow your program to enter an arbitrary number of terms before adding, such as 55+76+345=.

Allow your program to continue adding terms to the previous sum, as a calculator actually does. In other words, after entering 75+32=, you can then enter +45= and see the result 152.

Allow other operations, such as subtraction, multiplication, division, and exponentiation. The easiest way to do this is with the operators evaluated from left to right. A more sophisticated approach uses the algebraic hierarchy of operations. Or even parentheses.

Allow real arithmetic.

Math 287 Project 5
Permutations

due Nov. 5, 1987

For this project you are to write a program which asks the user for a positive integer N and then generates all permutations of the first N letters of the alphabet. For example, if the user enters 3, the program should print

ABC
ACB
BAC
BCA
CAB
CBA

The program should also count the number of permutations generated (6 in this case). You know that the answer in general is N!, but don't write a factorial function; have your program count the permutations as they are generated, so you can verify that all permutations were counted.

Your program must use recursion to generate the permutations. This is a natural method of solution. After all, it is easy to generate the permutations of one letter. Further, if you want to generate the permutations of N letters, you can take each letter in turn and "hire an assistant" to generate all permutations of the remaining N-1 letters. The permutations must be printed in alphabetical order, which they will be automatically if your program is written in an orderly way.

As often happens with recursion, the final program should be fairly short. In fact, my program was shorter than any of my programs for the previous four projects. On the other hand, the amount of thought per line for a recursive program is often very high.

Turn in a listing of your program along with the output for N = 1, 2, 3, 4, and 5.

Extra credit options:

I can't think of any. Got any ideas?

NOTES TO ACCOMPANY THE FILM 'SORTING OUT SORTING'

INSERTION: NEW ELEMENT PLACED INTO ORDERED DATA
****

LINEAR INSERTION
SORT THE FIRST TWO ITEMS. EACH ITEM AFTER THAT IS SORTED INTO PRE-ORDERED LIST SEQUENTIALLY. ORDER $\mathcal{O}(N^2)$.

BINARY INSERTION
POSITION OF NEW ITEM IN ORDERED DATA IS FOUND THROUGH A BINARY SEARCH (CUT DATA IN HALF TO SPEED UP THE SEARCH).
* SLOW GOING IF DATA IS PRETTY WELL SORTED ALREADY. ORDER $\mathcal{O}(N^2)$.

SHELL-METZNER SORT
WORK ON SMALL SUB-ARRAYS OF ITEMS PLACED FAR APART (TO MAXIMIZE ADVANCEMENT TO CORRECT LOCATION). THE SORT WITHIN EACH SUBARRAY IS BY INSERTION (COULD BE EXCHANGE OR SELECTION). SUCCESSIVE SUBARRAYS USE ITEMS CLOSER TOGETHER (SPAN:= SPAN DIV 2, WORKS BEST IF SUCCESSIVE SIZES ARE RELATIVELY PRIME). FINAL PASS IS ON ADJACENT ITEMS. ORDER $\mathcal{O}(N(\log N)^2)$.

EXCHANGE: EXCHANGE PAIRS OF ITEMS UNTIL ALL ARE IN ORDER
****

BUBBLE SORT
EXCHANGE ADJACENT PAIRS TO MOVE SMALLEST TO TOP. SUCCESSIVE PASSES MOVE SMALLEST OF REMAINING ITEMS TO EACH APPROPRIATE POSITION. ORDER $\mathcal{O}(N^2)$.

SHAKER SORT
'COCKTAIL SHAKER'. EXCHANGE PAIRS TO GET SMALLEST TO TOP THEN BRING LARGEST TO BOTTOM ON THE RETURN. CONTINUE WITH NEXT SMALLEST, NEXT LARGEST, ETC.
* 'SMART' - IT STOPS WHEN A PASS DOES NO SWAPS. ORDER $\mathcal{O}(N^2)$.

QUICK SORT
DEVELOPED BY HOARE. PIVOT CHOSEN AND ITEMS ARE COMPARED TO THE PIVOT UNTIL ALL ITEMS > PIVOT ARE ON ONE SIDE AND ALL ITEMS < PIVOT ARE ON THE OTHER SIDE. NEW PIVOTS ARE CHOSEN FOR THESE SMALLER SETS OF ITEMS AND THE PROCESS CONTINUES RECURSIVELY. ORDER $\mathcal{O}(N(\log N))$.

SELECTION: SELECT KEY ITEM AND MOVE IT INTO PLACE
****

STRAIGHT SELECTION
LINEAR SEARCH FOR SMALLEST ITEM, MOVE IT INTO PLACE. SEARCH FOR NEXT SMALLEST, ETC. ORDER $\Theta(N^2)$.

TREE SELECTION
PUT DATA AT BOTTOM OF A TREE. 'PROMOTE' THE SMALLER ITEM IN EACH PAIR UNTIL SMALLEST OF ALL IS AT THE TOP. PICK OFF THE TOP ITEM AND STORE IT IN AN ARRAY. PROMOTE THE NEXT SMALLEST TO THE TOP, PICK IT OFF AND STORE IT, ETC. 'PETER PRINCIPLE' OF AUTOMATIC PROMOTION.
* NEEDS A GREAT DEAL OF MEMORY SPACE (ORIGINAL ARRAY, TREE, FINAL ARRAY). ORDER $\Theta(N(\text{LOG } N))$.

HEAP SORT
DATA IS DISTRIBUTED IN A TREE. ITEMS MUST BE ARRANGED TO PLACE THE LARGEST VALUE FROM EACH SUBTREE AT ITS TOP (A 'HEAP'). THE LARGEST OF THE WHOLE TREE ('ROOT') IS THEN SWAPPED TO THE END POSITION AND THE TREE IS RE-HEAPED.
* BETTER MEMORY USE THAN TREE SELECTION. RUNS FAST. ORDER $\Theta(N(\text{LOG } N))$.

A COMPARISON OF THE RELATIVE MERITS OF SOME SORTS USING 1000 ITEMS
****

TYPE	COMPARISONS	SWAPS OR MOVES	TIME IN SECONDS
BUBBLE	499,479	242,428 SWAPS	3,538
LINEAR INSERTION	25,870	243,421 MOVES	2,060
BINARY INSERTION	11,496	243,623 MOVES	1,045
SHELL-METZNER	14,160	6,711 SWAPS	100
QUICK SORT	13,444	2,639 SWAPS	47

```
program showfunctions (input, output);
{demonstrates some funtions from sections 3.1, 3.2,3.3, and 3.7 of}
{"How to Solve it by Computer" by R.G. Dromey}

 function sqroot (a : real) : real;
{find the square root of a}
  const
   error = 1.0e-8;{error tolerated in the answer}
  var
   g1,       {previous estimate of square root}
   g2        {current estimate of square root}
    : real;
 begin
  g2 := a / 2;
  repeat
   g1 := g2;
   g2 := (g1 + a / g1) / 2;
  until abs(g1 - g2) < error;
  sqroot := g2
 end;{function sqroot}

 function smalldivisor (n : integer) : integer;
{finds the smallest exact divisor of an integer n}
  var
   d,       {current divisor and member of odd sequence}
   r :      {biggest integer <= sqrt(n)}
   integer;
 begin
  if not odd(n) then
   smalldivisor := 2
  else
   begin{search for odd divisor}
    r := trunc(sqrt(n));
    d := 3;
    while (n mod d <> 0) and (d < r) do
     d := d + 2;
    if n mod d = 0 then
     smalldivisor := d
    else
     smalldivisor := 1
   end{search for odd divisor}
end;{function smalldivisor}
```

```
  function gcd (n, m : integer) : integer;
 {find the greatest common divisor of two positive integers m and n}
   var
    r   {remainder after division of n by m}
     : integer;
  begin
   repeat
    r := n mod m;
    n := m;
    m := r;
   until r = 0;
   gcd := n
  end;{function gcd}

  function power (x, n : integer) : longint;
 {raise x to the n power}
   var
    product,     {current accumulated product}
    psequence :     {current power sequence value}
    longint;
  begin
   product := 1;
   psequence := x;
   while n > 0 do
    begin
     if (n mod 2) = 1 then
      product := product * psequence;
     n := n div 2;
     psequence := psequence * psequence
    end;
   power := product
  end;{function power}

begin{main program}
 writeln('The square root of 2 is ', sqroot(2) : 12 : 8);
 writeln('The smallest divisor of 901 is ', smalldivisor(901) : 1);
 write('The greatest common divisor of 1008 and 270 is ');
 writeln(gcd(1008, 270) : 1);
 writeln('3 raised to the 13th power is ', power(3, 13) : 1);
end.
```

```
program countem (input, output);
{ex. 8-14 of Oh! Pascal!}
{Count the number of words and sentences in some text}
 var
   symbol : char;{the latest symbol read}
   numwords, numsent : integer;{the number of words and sentences}
   inaword : boolean; {tells whether we are currently in a new word}

begin
 numwords := 0;
 numsent := 0;
 inaword := false;
 while not eoln do
  begin
   read(symbol);
   if symbol in ['.', '!', '?'] then
    begin{found the end of a sentence}
     numsent := numsent + 1;
     if inaword then
      begin {this is also the end of a word}
       numwords := numwords + 1;
       inaword := false;
      end; {of word at end of sentence}
    end{found the end of a sentence}
   else if (symbol <> ' ') and (not inaword) then
    inaword := true {found the beginning of a word}
   else if (symbol = ' ') and inaword then
    begin {found the end of a word}
     numwords := numwords + 1;
     inaword := false;
    end;{found the end of a word}
  end;{while not eoln}
 writeln;
 write('There are ', numwords : 1, ' words and ', numsent : 1, ' sentences');
 writeln(' in this text.');
end.
```

```
program magician (input, output);
{exercise 6-31 of Oh! Pascal, 2nd ed.}
{After doing some funny computations, the original number is}
{arrived at.}
{Written by Ray Greenwell, summer 1987, to thrill his Math 287 students.}
 var
  num,                          {the original number}
  a, b, c,                      {the digits of the original number}
  x, y, z, sum1, sum2, sum3 :   {some intermediate variables}
  integer;

 function resultmod11 (p, q, r : integer) : integer;
{returns the remainder of pqr, interpreted as a 3 digit number,}
{when divided by 11.}
 begin
  resultmod11 := (100 * p + 10 * q + r) mod 11
 end;{function resultmod11}

 procedure checkodd (var sum : integer);
{if the sum is odd, increase or decrease it, whichever results}
{in a nonnegative number less than 20}
 begin
  if odd(sum) then
   if sum < 11 then
    sum := sum + 11
   else {sum is odd and >=11}
    sum := sum - 11
 end;{procedure checkodd}

begin {main program}
 write('Enter a 3 digit number: ');
 readln(num);
 a := num div 100;{hundreds digit}
 b := num div 10 mod 10;{tens digit}
 c := num - a * 100 - b * 10;{ones digit}
 x := resultmod11(a, b, c);
 y := resultmod11(b, c, a);
 z := resultmod11(c, a, b);
 sum1 := x + y;
 sum2 := y + z;
 sum3 := z + x;
 checkodd(sum1);
 checkodd(sum2);
```

```
  checkodd(sum3);
  write('The result of the magic computation is ');
  writeln((sum1 div 2) : 1, (sum2 div 2) : 1, (sum3 div 2) : 1);
end.
```

```
program hpcalculator (input, output);
{Program to simulate a Hewlett Packard calculator.}
{Written for Math 287 by Ray Greenwell, Fall 1986}
 const
  maxsize = 10;
 type
  stacktype = record
    top : integer;
    entry : array[1..maxsize] of integer;
   end; {stacktype}
 var
  stack : stacktype;
  number, num1, num2 : integer;
  symbol : char;
  operators : set of char;

 function empty (stack : stacktype) : boolean;
{check to see if the stack is empty}
 begin
  if stack.top = 0 then
   empty := true
  else
   empty := false;
 end; {function empty}

 procedure push (var stack : stacktype;
           value : integer);
 begin
  with stack do
   if top >= maxsize then
    writeln('Stack overflow. Last entry ignored.')
   else
    begin
     top := top + 1;
     entry[top] := value
    end
end;{procedure push}

procedure pop (var stack : stacktype;
          var value : integer);
begin
 if empty(stack) then
  writeln('Stack empty.  Operator ignored.')
```

```
    else
     with stack do
      begin
       value := entry[top];
       top := top - 1
      end
  end; {procedure pop}

  function convert (symbol : char) : integer;
{convert a digit character to its numeric value}
  begin
   convert := ord(symbol) - ord('0');
  end;{function convert}

  function operate (num1, num2 : integer;
            operator : char) : integer;
{operate on num1 and num2 with the operator}
  begin
   case operator of
    '+' :
     operate := num2 + num1;
    '-' :
     operate := num2 - num1;
    '*' :
     operate := num2 * num1;
   end;{case}
  end;{function operate}

begin {main}
 operators := ['+', '-', '*'];
 writeln('Enter an expression in Reverse Polish Notation.');
 writeln('If you type an "r", the result will be printed.');
 writeln('Type a "q" to quit.');
 writeln('For simplicity, only one digit numbers may be entered,');
 writeln('and the only operators are +, -, and *.');
repeat
 read(symbol);
 if (symbol >= '0') and (symbol <= '9') then
  begin {digit case}
   number := convert(symbol);
   push(stack, number);
  end{digit case}
 else if symbol in operators then
```

```
      begin{operator case}
        pop(stack, num1);
        pop(stack, num2);
        push(stack, operate(num1, num2, symbol));
      end{operator case}
    else if symbol = 'r' then
      begin {print result case}
        writeln;
        if stack.top > 0 then
          writeln('Result= ', stack.entry[stack.top] : 1)
        else
          writeln('Stack is empty; no result.')
      end{print result case}
  until symbol = 'q';
  writeln;
  if stack.top > 0 then
    writeln('Result= ', stack.entry[stack:top] : 1);
end.
```

Math 287 Halloween Midterm

name Blaise Pascal

October 30, 1986

Worth 130 points.

In all mathematical problems, be sure to show all your work.

1) (11 pts) Compute 151^{151} to the accuracy of your calculator. And don't tell me your calculator won't compute a number that big; I already know that.

$151^{30} = 2.340498778 \times 10^{65}$

$151^{151} = (2.340498778 \times 10^{65})^5 \times 151$

$= 10605.20433 \times 10^{325}$

$= 1.060520433 \times 10^{329}$

or $10^{151 \log_{10} 151} = 10^{151 \times 2.178976947}$

$= 10^{329.025519}$

$= 10^{329} \times 10^{.025519} = 1.060520383 \times 10^{329}$

2) (16 pts) In the quadratic formula problem done in class, we computed DISCR:= B*B - 4*A*C, and if this quantity was negative, we printed out the message "No real roots." Now you are to modify the program so that if the roots are, for example, 3 ± 2i, the program will print "The complex conjugate roots are 3.0 + 2i and 3.0 - 2i." All you need do is write the Pasca statements that go between the **begin** and **end** in the following statement

```
if discr < 0 then
  begin
    :
    :
  end
```

```
REALPART:= -B/(2*A);
IMAGPART:= SQRT(-DISCR)/(2*A);
WRITE('The complex conjugate roots are ');
WRITELN(REALPART:3:1,'+',IMAGPART:3:1,' and ',REALPART:3:1,'-',
        IMAGPART:3:1,'.');
```


BEST COPY AVAILABLE

3) (13 pts) Write a procedure that adds together two matrices A and B and puts the result in C. For example, if A = $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \end{bmatrix}$ and B = $\begin{bmatrix} 0 & 2 & -1 \\ 3 & 1 & 2 \end{bmatrix}$, then C = $\begin{bmatrix} 1 & 4 & 2 \\ 3 & 2 & 4 \end{bmatrix}$.

You may assume that the program contains the statement
type matrix = **array**[1..10, 1..10] of **integer**;
Letting "rows" and "cols" be the number of rows and columns in the matrices, here is the first statement of your procedure:

```
procedure addmat(var A, B, C: matrix; rows, cols: integer);
VAR I,J: INTEGER;
 BEGIN
 FOR I:=1 TO ROWS DO
    FOR J:=1 TO COLS DO
        C[I,J]:=A[I,J]+B[I,J];
 END;{procedure addmat}
```

4) (12 pts) Suppose 16! (that is: 16·15·14·...·3·2·1) is written as a binary number. How many trailing 0's are there, counting from the right, until the first 1 is encountered?

In base 2, every factor of 2 contributes to a 0, just as in base 10 every 5 (combined with a 2) contributes to a 0.

$16 \cdot 15 \cdot 14 \cdot 13 \cdot 12 \cdot 11 \cdot 10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$

$2^4 \quad 2 \quad 2^2 \quad 2 \quad 2^3 \quad 2 \quad 2^1 \quad 2$

Total # of 2's = 4+1+2+1+3+1+2+1 = 15

There are 15 trailing 0's.

5) (18 pts) Use a Boolean variable to rewrite the following section of Pascal code without using any **goto**'s. The comments in brackets refer to unspecified sections of code.

```
for i:=1 to n do
    begin
    {stuff}
    if x < 0 then goto later;
    {more stuff}
    if y >= 0 then goto later;
    {even more stuff}
    end;
later: {the rest of the program goes here};
```

```
VAR DONE : BOOLEAN
  ;

I := 1;
DONE := FALSE;
WHILE (I <= N) AND (NOT DONE) DO
    BEGIN
    {stuff}
    IF X<0 THEN DONE := TRUE
           ELSE BEGIN
                {more stuff}
                IF Y>=0 THEN DONE := TRUE
                        ELSE BEGIN
                             {even more stuff}
                             I := I+1;
                             END; {second ELSE}
                END; {first else}
    END; {while}
{the rest of the program goes here}
```

6) (16 pts) Suppose we represent fractions by the type

```
type fraction = record
                    numerator, denominator: integer
                end;
```

Write a procedure which adds the fractions A and B to give the result C. You may assume that A and B are already reduced, but you need not reduce C. However, you must find use the lowest common denominator of A and B when finding C. For example, if A = 3/10 and B = 7/15, C should equal (3/10)*(3/3) + (7/15)*(2/2) = 23/30. You may assume that we already have created a function LCM which gives the least common multiple of two numbers (e.g. LCM(10, 15) = 30). Here is the first line of the procedure:

```
procedure addfrac(var A, i, C: fraction);
VAR AFACTOR, BFACTOR : INTEGER;
BEGIN
C.DENOMINATOR := LCM (A.DENOMINATOR, B.DENOMINATOR);
AFACTOR := C.DENOMINATOR DIV A.DENOMINATOR;
BFACTOR := C.DINOMINATOR DIV B.DENOMINATOR;
C.NUMERATOR := AFACTOR * A.NUMERATOR + BFACTOR * B.NUMERATOR;
END; { procedure addfrac }
```

7) (18 pts) If $a_1, a_2, a_3, \ldots$ is a sequence of real numbers such that $a_{n+1} = (n*a_n - 1)/(n+1)$, find a closed formula for a_n (as a function of n and a_1) and prove this formula by induction. Then find $\lim_{n\to\infty} a_n$.

$$a_2 = \frac{1\cdot a_1 - 1}{1+1} = \frac{a_1 - 1}{2}$$

$$a_3 = \frac{2\cdot a_2 - 1}{2+1} = \frac{(a_1 - 1) - 1}{3} = \frac{a_1 - 2}{3}$$

$$a_4 = \frac{3\cdot a_3 - 1}{3+1} = \frac{(a_1 - 2) - 1}{4} = \frac{a_1 - 3}{4}$$

Claim: $a_n = \frac{a_1 - (n-1)}{n}$

PROOF: True for $n = 1, 2, 3, 4$ by above calculations

Assume: $a_k = \frac{a_1 - (k-1)}{k}$

Prove: $a_{k+1} = \frac{a_1 - k}{k+1}$

$$a_{k+1} = \frac{k\,a_k - 1}{k+1} \quad \text{by original formula}$$

$$= \frac{(a_1 - (k-1)) - 1}{k+1} \quad \text{by assumption}$$

$$= \frac{a_1 - k}{k+1} \quad \text{done!}$$

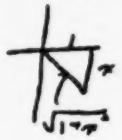
$$\lim_{n\to\infty} a_n = \lim_{n\to\infty} \frac{a_1}{n} - 1 + \frac{1}{n} = -1$$

($\frac{a_1}{n} \to 0$, $\frac{1}{n} \to 0$)

8) Terr'bl Pascal is a non-standard version of Pascal, available at Waldbaums for $3.95, or available for free with 17 boxtops from Kellogg's Raisin Bran.

a) (14 pts) The only inverse trigonometric function in Terr'bl Pascal is **arcsec**(x), which gives $\sec^{-1}x$ in radians if $x \geq 1$ and bombs for any other values of x. Use this function to define an **arctan** function that will give $\tan^{-1}x$ in radians for <u>all</u> real numbers x. Here is the first line:

x negative:




```
function arctan(x. real): real;
BEGIN
IF X >=0  THEN ARCTAN:= ARCSEC(SQRT(1 + X*X))
          ELSE ARCTAN:= -ARCSEC(SQRT(1+ X*X))
END; {function arctan}
```

b) (12 pts) Terr'bl Pascal has the **div** function of standard Pascal, so p **div** q gives the integer quotient of p/q, where p and q are integers. Unfortunately, it has no **mod** function, which gives the remainder. Create such a function, so that, for example, mod(5,17) returns the value 2.

```
FUNCTION MOD(A, B: INTEGER): INTEGER;
BEGIN
MOD:= A - (A DIV B) * B
END; {function mod}
```

Math 287 Exciting Midterm

name Blaise Pascal

Worth 110 yummy points.

July 30, 1987

1) (15 pts) The following program is supposed to take the average of a set of nonnegative numbers and print the average to 2 decimal places. A negative number is to be read in at the end of the set; it is not to be counted in the average. Unfortunately, the program was written by my untrusty assistant igor and contains some errors. Correct all the errors, but do not change anything that is correct.

```
program matic(input,output);
var sum, count, value, average: integer;        <- AVERAGE : REAL;
begin
count:=0;
value:=0;
while value>=0 do
   begin
   sum:=0;
   read(value);
   sum:=sum + value;
   count:=count + 1;                            <- READ ( VALUE);
   end;
average:=sum/count;                             IF COUNT > 0
writeln('The average is ',average:4:2);             THEN BEGIN
end.                                                    AVERAGE := SUM/COUNT;
                                                        WRITELN('The average is', AVERAGE:4:2)
                                                        END
                                                ELSE WRITELN('No numbers entered.');
```

2) (15 pts) The latest version of Terr'bl Pascal has an arccos function built in but no arctan function. Use the arccos function to write a function that returns the arctan of any real number x. Here is the first line:

function arctan(x: real): real;

```
BEGIN
IF X>=0  THEN ARCTAN:=ARCCOS(1/SQRT(1+X*X))
         ELSE ARCTAN:=-ARCCOS(1/SQRT(1+X*X))
END;
```

3) (15 pts) Explain briefly what the following program accomplishes, assuming it reaches a successful conclusion without running into integer overflow problems.

```
program others(input,output);
var a,b,c: integer;
begin
a:=0;
b:=0;
c:=0;
while b < 3 do
   begin
   c:=c+1;
   a:=a + sqr(c);
   if sqr(round(sqrt(a)))=a
      then begin
           writeln(a);
           b:=b+1;
           end;
   end;
end.
```

$A = 1^2+2^2+\ldots+N^2$

Everytime a value of A is found that is a perfect square, B is increased by 1. We leave the loop once B=3.

Thus, this program writes out the first three positive integers that are perfect squares as well as sums of the form $1^2+2^2+\ldots+N^2$.

BEST COPY AVAILABLE

4) (15 pts) What is the output of the following procedure?

```
procedure a1;
var c: integer;
begin
for c:=10 to 20 do
   case c*3 mod 5 of
      0,4: write('who ');
      1:  write('what ');
      2:  write('sees ');
      3:  write('knows ');
   end;
end;
```

WRITELN

c	c*3 MOD 5
10	30 → 0
11	33 → 3
12	36 → 1
13	39 → 4
14	42 → 2
15	45 → 0
16	48 → 3
17	51 → 1
18	54 → 4
19	57 → 2
20	60 → 0

output {
who knows what
who sees who knows what
who sees who
}

5) (15 pts) Write a Pascal function with the single parameter NUMBER, assumed to be a three-digit integer. The function should return an integer that contains the digits in reverse order.

```
FUNCTION REVERSE (N: INTEGER): INTEGER;
VAR HUND, TENS, ONES : INTEGER;
BEGIN
ONES:= N MOD 10;
TENS: = N DIV 10 MOD 10;
HUND: = N DIV 100 MOD 10;
REVERSE: = ONES*100 + TENS * 10 + HUND
END;
```

6) (15 pts) Suppose the following input is entered into the program below:
If McDougal programs in FORTRAN, then d'Antonio programs in Pascal.
Assuming that all **write**s are sent to the printer, but what is **read** only appears on the screen, what is printed by the program?

```
program ming(input,output);
var ch: char;
begin
repeat                                  <- keep reading (and writing) characters until a period is found.
   read(ch);
   if ch in ['A'..'Z']                  <- If a capital letter is found, stay in this section
      then repeat                          until a space or period is found.
              write(ch);
              read(ch);
              if ch in ['a'..'z']
                 then ch:=chr(ord(ch) + ord('A') - ord('a'));   } If a lower case letter is found in this section, convert it to upper case.
              until (ch=' ') or (ch='.');
   write(ch);
   until ch='.';
end.
```

output { IF MCDOUGAL programs in FORTRAN, then d'ANTONIO programs in PASCAL.

7) (20 pts) Use a Boolean variable to rewrite the following section of Pascal code without any **goto**'s. The comments in brackets refer to unspecified sections of code.

```
for i:=n downto m do
   begin
   {stuff}
   for j:=i to n do
      begin
      {more stuff}
      if x=0 then goto later;
      {even more stuff}
      end;
   {still more stuff}
   end;
later: {rest of program follows}
```

```
VAR DONE: BOOLEAN;
 :
I:=N;
DONE:=FALSE;
WHILE (NOT DONE) AND (I>=M) DO
   BEGIN
   {stuff}
   J:=I;
   WHILE (J<=N) AND (NOT DONE) DO
      BEGIN
      {more stuff}
      IF X=0 THEN DONE:=TRUE
             ELSE BEGIN {even more stuff}
                  J:=J+1;
                  END; {ELSE}
      END; {WHILE J...}
   IF NOT DONE
      THEN BEGIN
           {still more stuff}
           I:=I+1;
           END; {THEN}
   END; {while I}
{rest of program follows}
```

should be lined up here.

BEST COPY AVAILABLE

And now, what you've been waiting all semester for,

MATH 287 FINAL EXAM

Dec. 16, 1966 Worth 144 points. name_____

Part 1. Multiple choice. (worth 60 points)
4 pts for each correct answer, -1 pt for each incorrect answer. No penalty for those left blank.

1. Suppose that one of many tasks a large program must perform is to display information about particular items in a table. (Some of the other tasks also involve searching and manipulating the table and displaying values.) Of the following, which indicates the best design of a procedure P that performs this task upon receiving a specification for an item?

☐a) P searches the table for the specified item and then displays the information about that item.

☐b) P displays the whole table.

☐c) P calls a procedure S to search the table for the specified item and then calls a procedure D to display the information about that item.

☐d) P searches the table for the specified item and then calls a procedure D to display the information about that item.

☐e) P calls a procedure S to search the table for the specified item and then itself displays the information about that item.

2) Suppose that items X1, X2, X3, X4, and X5 are pushed, in that order, onto an initially empty stack S, that S is then popped four times, and that as each Xi is popped off S, that Xi is then inserted into an initially empty queue. If one Xi is then deleted from the queue, what is the <u>next</u> item that will be deleted from the queue?

☐a) X1 ☐b) X2 ☐c) X3 ☐d) X4 ☐e) X5

3) A hypothetical digital computer computes the results of the four ordinary binary operations +, -, *, / to 4 significant figures and returns the result of each operation truncated to 3 significant figures. If the expression
(6.74 - 0.024)*2.00 - 8.38
is used as input, what is the result returned by this computer?

☐a) 5.00 ☐b) 5.02 ☐c) 5.03 ☐d) 5.04 ☐e) 5.06

Questions 4 and 5 are based on a program with the following structure:

```
program A;
  procedure B;
    procedure C;
    begin
      ...
      C;
      ...
    end;
  procedure D;
    begin
      ...
      B;
      ...
      B;
      ...
    end;
  begin
    ...
    D;
    ...
  end.
```

4) A variable that is declared in procedure B and only in procedure B is accessible in

☐a) all of program A

☐b) procedure B, but not in procedures C and D

☐c) procedures B and C, but not in procedure D

☐d) procedures B and D, but not in procedure C

☐e) procedures B, C, and C, but not elsewhere in A

5) Suppose that the program A contains no **goto** statements and no procedure calls other than those indicated. Which of the following lists describes completely the order in which the procedures conteined in program A are called or invoked?

☐a) D, B, C

☐b) B, C, D

☐c) C, B, B, D

☐d) D, B, B, C

☐e) D, B, C, B, C

6) Which of the following statements, when used as the body of the function definition **procedure** Fact(n:integer) : integer;

```
begin
  .
  .
end;
```

will enable that function to compute n! correctly for any n > 0, where n! = n*(n-1)*(n-2)* . . .*3*2*1 ?

I. Fact := n*Fact(n-1)

II. **if** n < 2 **then** Fact := n
else Fact := n*Fact(n-1)

III. **if** n = 1 **then** Fact := 1
else Fact := Fact(n+1)/(n+1)

☐a) I only

☐b) II only

☐c) I and II only

☐d) II and III only

☐e) I, II, and III

7) What output is produced by the following program?

```
program ABC(input, output);
    var n: integer;

    procedure Increment(var a, b: integer);
    begin
        a:= a+1;
        b:= b+1
    end;

    begin
        n:= 2;
        Increment(n,n);
        write(n)
    end.
```

☐a) 5 ☐b) 4 ☐c) 3 ☐d) 2 ☐e) An error message

8) Consider the following sequence of procedure calls:

```
Push(x);
Push(y);
Add;
Push(z);
Push(w);
Mult;
Add;
```

Invoking Push causes its argument to be pushed onto a stack. Invoking the procedures Add or Mult causes (1) the stack to be popped twice, (2) the two popped items to be added or multiplied, and (3) the result to be pushed onto the stack. If x = 20, y = 30, z = 20, and w = 70, and the stack is empty initially, then at the end of the sequence of procedure call above, the stack contains

☐a) Nothing ☐b) 0 ☐c) 1070 ☐d) 1230 ☐c) 1450

9) If there are 10 internal nodes (that is, nodes that are not leaves) in a binary tree, at most how many leaves can there be?

☐a) 1 ☐b) 4 ☐c) 9 ☐d) 11 ☐e) 20

Questions 10 and 11 are based on the following declarations.

```
type SexType    = (M, F);
     PartyType  = (Christmas, Halloween, Comeasyouare, Other);
     AgeType    = 0..125;
   CitizenType  = record
                    Sex   : SexType;
                    Party : PartyType
                    Age   : AgeType;
                  end;
var Citizen   : CitizenType;
    Longevity : AgeType;
```

10) Which of the following is a valid Pascal statement?

☐a) writeln(Citizen)

☐b) writeln(Citizen.Sex)

☐c) writeln(Citizen.Age)

☐d) writeln(Age.Citizen)

☐e) writeln(CitizenType.Age)

11) If G is a function with header

```
function G(x, y: AgeType) : AgeType;
```

which of the following is a valid Pascal statement?

☐a) G

☐b) G(1,3)

☐c) read(G(1,3))

☐d) Longevity:= G(5, Citizen.Party)

☐d) writeln(G(70,Longevity))

12) Suppose that the following program segment is used to approximate a zero of the real-valued function

f(x) = x*x - 3

starting with Left = 1 and Right = 2.

```
var Left, Right, x, Epsilon: real;
repeat
   x:= (Left + Right)/2;
   if f(x) < 0 then  Left:= x
               else Right:= x
until (Right - Left) < Epsilon
```

How many times must the loop be executed to produce an x that is guaranteed to be within Epsilon of a zero of f when Epsilon is 0.001?

☐a) Once

☐b) 10 times

☐c) 100 times

☐d) 1000 times

☐e) It cannot be determined from the information given.

13) Let Rnd be a function that returns a random value uniformly distributed between 0 and 1. Which of the following expressions, when inserted at the indicated point in the program segment

```
Count:= 0;
for i:= 1 to n do
   if (expression goes here) then
      Count:= Count + 1;
Pi:= 4*Count/n
```

will cause that program segment to approximate the number Pi = 3.14159... by a Monte Carlo method?

I. sqr(Rnd) + sqr(Rnd) < 1

II. Rnd*Rnd + Rnd*Rnd < 1

III. Rnd + Rnd < 1

☐a) I only ☐b) II only ☐c) III only ☐d) I and II ☐e) I and III

14) A certain binary tree T is represented as a two-dimensional 5×3 array A, with rows of A corresponding to nodes of T. The columns of A contain the following information:

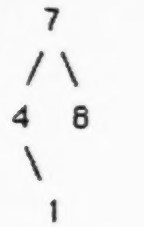
column 1--the matrix row index of the left child
column 2--the value stored at the node
column 3--the matrix row index of the right child

A matrix row index of 0 indicates a nonexistent child. If A consists of the entries

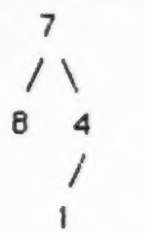
5	7	4
0	0	1
0	1	0
3	4	0
0	8	0

then T is given by which of the following diagrams?

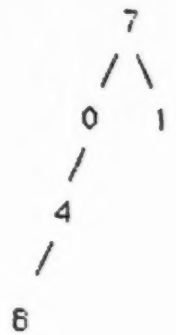
☐a)



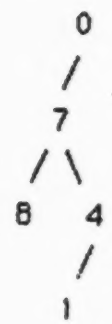
☐b)



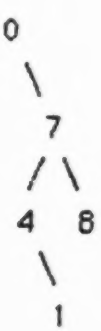
☐c)



☐d)



☐e)



15) Suppose that the variable d represents the number of dollars in a bank account after interest has just been credited to that account (e.g., d = 123.456). Which of the following Pascal expressions would round that amount to the nearest cent (e.g., to 123.46)?

☐a) round(d/100)*100

☐b) round(100*d)/100

☐c) round(d/100)

☐d) round(100*d)

☐e) round(d)

Part 2. Fun part. (worth 84 points)

1a) (8 pts) Represent the expression A/B - C*D^(E - G + G*H) as a binary tree.

b) (6 pts) Write the expression from part a) in postfix notation (i.e. Reverse Polish Notation).

c) (6 pts) Write the expression from part a) in prefix notation.

2) (9 pts) Here is Ackermann's function again:

$$A(m,n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m \neq 0, n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m \neq 0, n = 0 \end{cases}$$

Compute A(2,0). Show your work!

3) Consider the following Pascal function, defined for $N \geq 1$:

```
function f(N: integer): integer;
begin
if N = 1 then f:= 0
        else f:= N div 2 + f(N div 2 + N mod 2)
end;
```

a) (10 pts) Compute f(1), f(2), f(3) and f(4).

b) (5 pts) Based on your answer from part a), you should be able to guess a very simple, nonrecursive version of the function f. Write such a function. (Note: if an extremely simple function is not obvious from part a), you've done something wrong, and you won't be able to do part c). If you still want a shot at doing part c), bring your test up to me and I'll grade parts a) and b) immediately and give you the correct answers.)

c) (11 pts) Use induction to prove that you answer from part b) gives the exact same result as the original function in part a). (Hint: to compute f(K + 1), consider even and odd cases.)

4) (6 pts) Show how the postfix expression 4 3 2 * 5 7 + - * is evaluated with a stack by showing the contents of the stack at each step. (Note: all numbers in the expression are single digit positive numbers. This will not necessarily be true of all numbers in the stack.)

5) (12 pts) Here is a section of code from a hypothetical game, where at various points in the game, if more than 100 seconds have elapsed, the game is over. The comments refer to unspecified sections of code. Use a Boolean variable to write a section of code that performs exactly the same actions as the code below, but without use of any **goto**'s.

```
while citiesleft > 0 do
   begin
   {first bunch of stuff}
   if time > 100 then goto gameover;
   {second bunch of stuff}
   if time > 100 then goto gameover;
   {third bunch of stuff}
   end;
gameover: writeln('Game over!');
   :
   :
```

6) (11 pts) Assume we have the following declarations in the beginning of our program:

```
type smallset = set of char;
     class = set of smallset;
var hold: smallset;
    result: class;
    i1, i2: integer;
```

Trace through the following section of Pascal code, showing the values of the variables at each step. Tell what the final value of the set "result" is, and describe in words what this section of code accomplishes. (Note: actually, this would not work in standard Pascal, since the elements of sets must be of an enumerated type, so we cannot have types such as "class". Ignore this restriction while doing the problem, because to get around it would be rather complicated.)

```
hold:= [ ];
result:= [ ];
for i1:= 1 to 2 do
   begin
   if i1=1 then hold:= hold + ['a']
           else hold:= hold - ['a'];
   for i2:= 1 to 2 do
      begin
      if i2=1 then hold:= hold + ['b']
              else hold:= hold - ['b'];
      result:= result + hold
      end {for i2 loop}
   end; {for i1 loop}
```